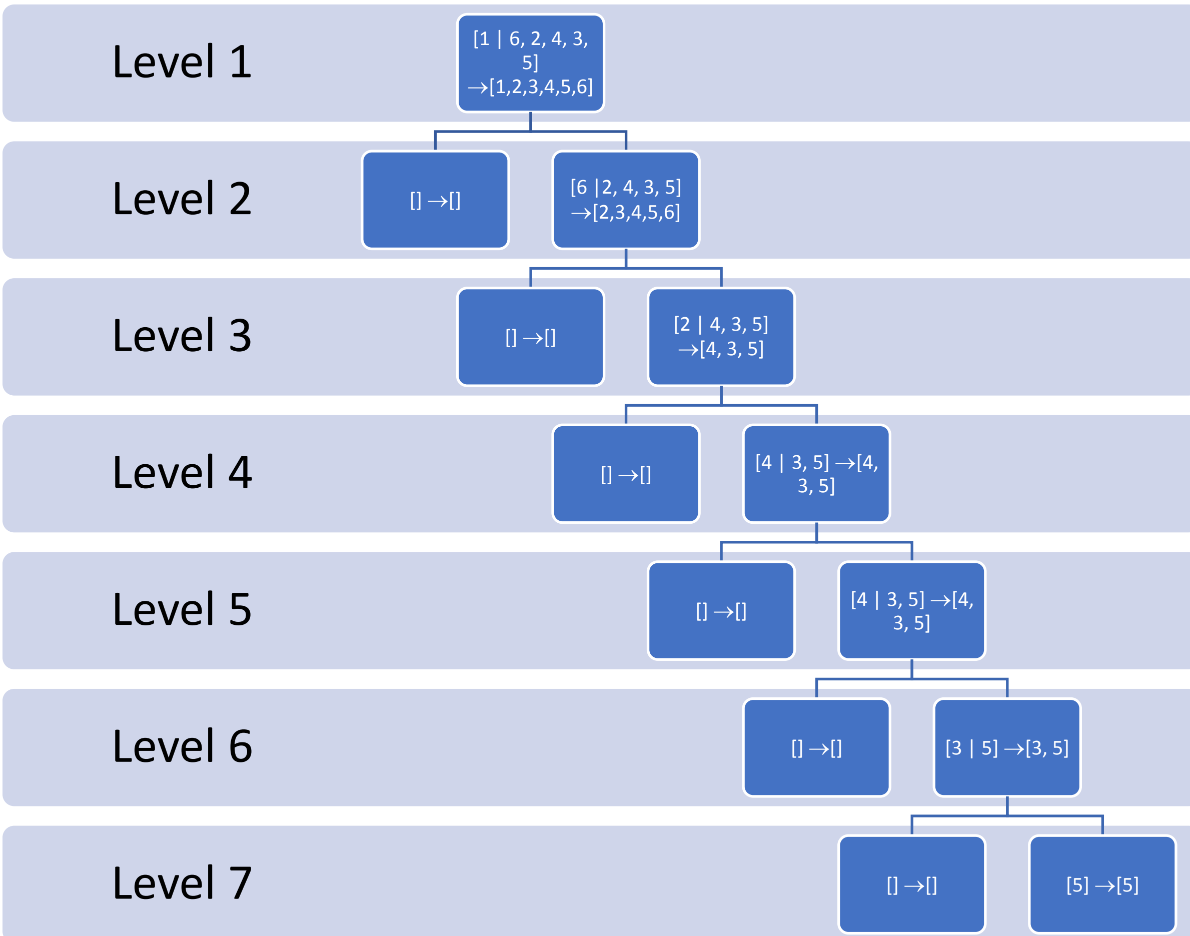
Algorithm: Lab5 (By Sujiv Shrestha ID:610145)
Problem 1.

1. Show all steps of QuickSort in sorting the array [1, 6, 2, 4, 3, 5]. Use leftmost values as pivots at each step.



Problem2

2. Show all steps of In-Place QuickSort in sorting the array [1, 6, 2, 4, 3, 5] when doing first partition. Use leftmost values as pivots.

Step 1: k=0

1	6	2	4	3	5
0	1	2	3	4	5

↑ **Pivot**

Step 2: Swap k^{th} element with rightmost element (r^{th})

5	6	2	4	3	1
0	1	2	3	4	5

↑ **Pivot**

Step 3: x = 1 (pivot element)

Step 4: in-place partition and get position of pivot point

a. Starts with (i=0 and j = r-1)

5	6	2	4	3	1
0	1	2	3	4	5
i				**j**	

b. i sticks at 0 as A[0]>pivot(1) and j follows till it crosses i, at j=-1

	5	6	2	4	3	1
-1	0	1	2	3	4	5
j	**i**					↑ **Pivot**

c. Swap pivot at r with i^{th} element.

1	5	6	2	4	3
0	1	2	3	4	5

↑ **Pivot**

Problem3

3. In our average case analysis of QuickSort, we defined a *good self-call* to be one in which the pivot *x* is chosen so that number of elements $< x$ is less than $3n/4$, and also the number of elements $> x$ is less than $3n/4$. We call an x with these properties a *good pivot.* When n is a power of 2, it is not hard to see that at least half of the elements in an n-element array could be used as a good pivot (exactly half if there are no duplicates). For this exercise, you will verify this property for the array A = [5, 1, 4, 3, 6, 2, 7, 1, 3] (here, n = 9). Note: For this analysis, use the version of QuickSort in which partitioning produces 3 subsequences *L*, *E*, *R* of the input sequence *S*.

a. Which x in A are good pivots? In other words, which values x in A satisfy:

i. the number of elements $< x$ is less than $3n/4$, and also

ii. the number of elements $> x$ is less than $3n/4$

Answer:

Input Array:

5	1	4	3	6	2	7	1	3
0	1	2	3	4	5	6	7	8

Input Array in sorted order:

Good pivot points

1	1	2	3	3	4	5	6	7
0	1	2	3	4	5	6	7	8

The good pivots that satisfy i. and ii. are [2,3,3,4,5]

b. Is it true that at least half the elements of A are good pivots?
Yes, it is true that at least half the elements of A are good pivots.

Problem4

4. *Interview Question.* Give an o(n) ("little-oh") algorithm for determining whether a sorted array A of distinct integers contains an element m for which A[m] = m. You must also provide a proof that your algorithm runs in o(n) time.

Solution:

```
Algorithm check(S, lower)
    Input sorted sequence S with n integers and number lower
    Output m if A[m]=m otherwise null
    mid←n/2
    if(n≤0) then
        return null
    if(S[mid]=mid+lower) then
        return S[mid]
    else if(S[mid]<mid+lower) then
        S1← S.copyRange(mid, n-1)
```

return check(S1, mid+lower)

else

S2← S.copyRange(0, mid-1)

return check(S2, lower)

Proof: In the worst case when m = 0 where A[m] = m, the number of recursive calls are equal to the number of terms in sequence S: n/2, n/4, n/8, ……………, $n/2^m$ (= 1) [where m = logn]. Hence, the running time for this algorithm in worst case is Θ(m) or Θ(logn). And we know that logn is o(n).

Problem5

5. Review of SubsetSum Problem: Given a set S = {s_0, s_1, s_2, …, s_{n-1}} of positive integers and a non-negative integer k, find a subset T of S so that the sum of the integers in T equals k or indicate no such subset can be found.

We have already seen a brute force solution to this problem in an earlier lab. In this exercise, you are going to come up with a recursive solution for SubsetSum. Write the pseudo code for your algorithm.

Hint:

We are seeking a $T \subseteq S = \{s_0, s_1, \ldots, s_{n-2}, s_{n-1}\}$ whose sum is k. Such a T can be found if and only if one of the following is true:

(1) A subset T_1 of $\{s_0, s_1, \ldots, s_{n-2}\}$ can be found whose sum is k, OR

(2) A subset T_2 of $\{s_0, s_1, \ldots, s_{n-2}\}$ can be found whose sum is $k - s_{n-1}$

If (1) holds, then the desired set T is T_1. If (2) holds, the desired set T is $T_2 \cup \{s_{n-1}\}$.

Solution:

Algorithm subsetSum(S, k)

Input sequence S with n positive integers and a non-negative integer k

Output subset T of S whose sum of elements is equal to k

if(k=0) **then**

return emptyList

else

for i← 0 to n-1 **do**

S1←S

p = S1.remove(i)

S2=subsetSum(S1, k-p)

if(S2 **is not** null) **then**

return {p}∪S2

return null

Java implementation using List:

```
public static List<Integer> subsetSum(List<Integer> a, int sum) {
        if(sum==0)
                return new ArrayList<Integer>();
        else{
                for(int i=0;i<a.size();i++) {
                        List<Integer> param = new ArrayList<>();
```

```java
				param.addAll(a);
				Integer p = param.remove(i);
				List<Integer> ans = subsetSum(param,sum-p);
				if(ans!=null) {
					ans.add(0, p);
					return ans;
				}
			}
			return null;
		}
	}
```

Java implementation using int array.

```java
	public static int[] subsetSum(int[] a, int sum) {
		if(sum==0)
			return new int[0];
		else{
			for(int i=0;i<a.length;i++) {
				int[] param = new int[a.length-1];
				System.arraycopy(a, 0, param, 0, i);
				System.arraycopy(a, i+1, param, i, a.length-i-1);
				Integer p = a[i];
				param = subsetSum(param,sum-p);
				if(param!=null) {
					int[] ans = new int[param.length+1];
					System.arraycopy(param, 0, ans, 1, param.length);
					ans[0] = p;
					return ans;
				}
			}
			return null;
		}
	}
```